

Git

Git

起步

- 下载

- 安装

- Bash基本命令

Git基础

- 文件的两种状态

- Git三种状态

Git配置

Git基本操作

- 获取Git仓库

 - 在已存在的目录中初始化仓库

 - 克隆现有的仓库

- 记录每次更新到仓库

 - 检查当前文件状态

 - 跟踪新文件

 - 暂存已修改文件

 - 状态简览

 - 忽略文件

 - 查看已暂存和未暂存的修改

 - 提交更新

 - 跳过使用暂存区

 - 移除文件

 - 重命名文件

- 撤销操作

- 远程仓库的使用

 - 查看远程仓库

 - 添加远程仓库

 - 从远程仓库抓取与拉取

 - 推送到远程仓库

 - 查看某个远程仓库

 - 远程仓库的重命名与移除

SSH

标签

- 列出标签

- 创建标签

 - 附注标签

 - 轻量标签

 - 后期打标签

 - 删除标签

Git别名

Git分支

起步

下载

[Git \(git-scm.com\)](https://git-scm.com/)

[git-for-windows Mirror \(taobao.org\)](https://github.com/git-for-windows/mirror)

安装

傻瓜式安装，更改安装目录，保持默认即可

Bash基本命令

命令	说明	备注
cd	改变目录	cd 目录; cd .. 回退到上级目录;
pwd	显示当前所在目录	
ls	列出当前目录的所有文件	ll 也是列出当前目录的所有文件，不过ll 显示的内容更详细
touch	新建文件	touch hello.java 在当前目录下创建一个hello.java文件
rm	删除文件	rm hello.java 将当前目录下的hello.java删除
mkdir	创建文件夹	mkdir test 创建一个test文件夹
rm -r	删除文件夹	rm -r test 删除test文件夹
mv	移动文件	mv hello.java src hello.java是要移动的文件，src是目标文件夹
clear	清屏	
history	查看命令历史记录	
help	帮助	
exit	退出	

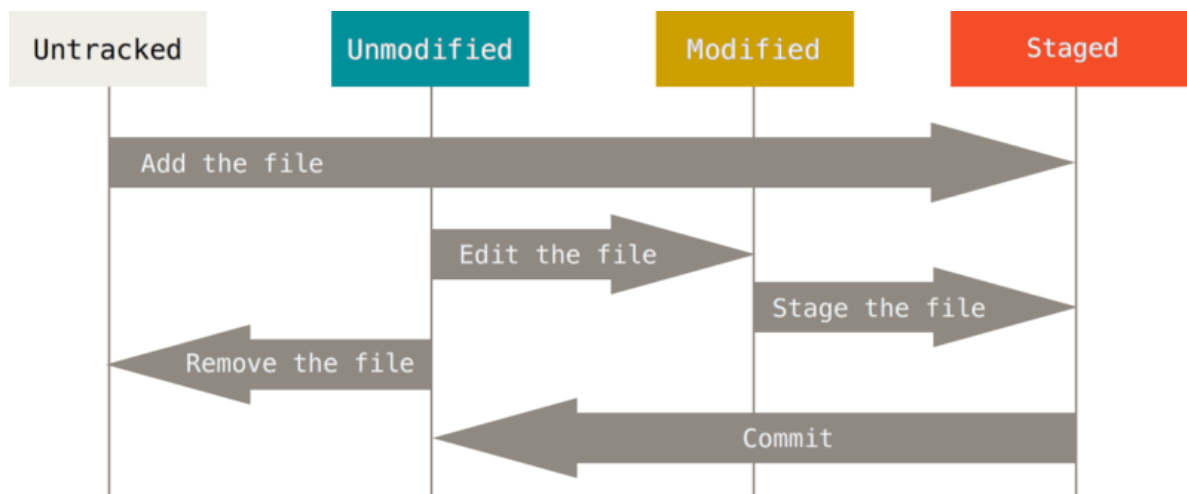
Git基础

文件的两种状态

- 请记住，你**工作目录**下的每一个文件都不外乎这两种状态：**已跟踪** 或 **未跟踪**。
 - 已跟踪的文件是指那些被纳入了版本控制的文件，在上一次快照中有它们的记录，在工作一段时间后，它们的状态可能是未修改（Unmodified），已修改（Modified）或已放入暂存区（Staged）。
 - 简而言之，已跟踪的文件就是 Git 已经知道的文件。
- 工作目录中除已跟踪文件外的其它所有文件都属于未跟踪文件，它们既不存在于上次快照的记录中，也没有被放入暂存区。

- 初次克隆某个仓库的时候，工作目录中的所有文件都属于已跟踪文件，并处于未修改状态，因为 Git 刚刚检出了它们，而你尚未编辑过它们。

编辑过某些文件之后，由于自上次提交后你对它们做了修改，Git 将它们标记为已修改文件。在工作时，你可以选择性地将这些修改过的文件放入暂存区，然后提交所有已暂存的修改，如此反复。



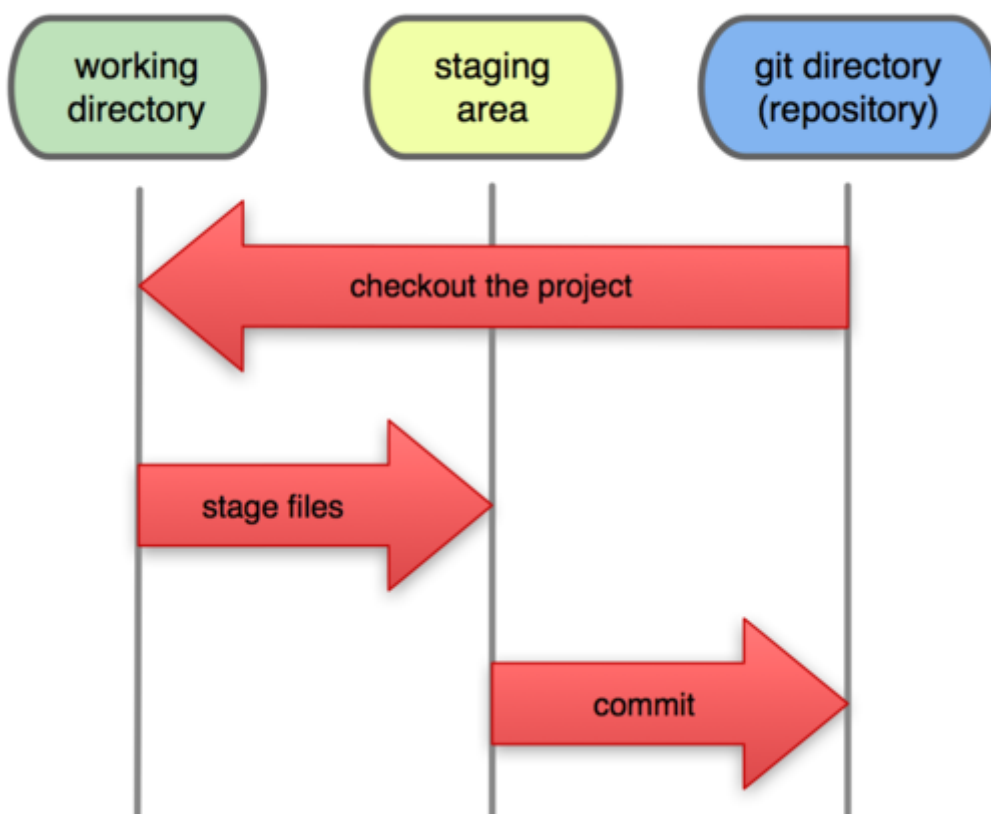
Git三种状态

对于任何一个文件，在 Git 内都只有三种状态：**已提交 (committed)**，**已修改 (modified)** 和**已暂存 (staged)**。

- 已提交表示该文件已经被安全地保存在本地数据库中了
- 已修改表示修改了某个文件，但还没有提交保存
- 已暂存表示把已修改的文件放在下次提交时要保存的清单中

由此我们看到 Git 管理项目时，文件流转的三个工作区域：Git 的工作目录，暂存区域，以及本地仓库。

Local Operations



基本的 Git 工作流程如下：

1. 在工作目录中修改某些文件。
2. 对修改后的文件进行快照，然后保存到暂存区域。
3. 提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中。

Git配置

• 必要配置

安装完 Git 之后，要做的第一件事就是设置你的用户名和邮件地址。这一点很重要，因为每一个 Git 提交都会使用这些信息，它们会写入到你的每一次提交中。

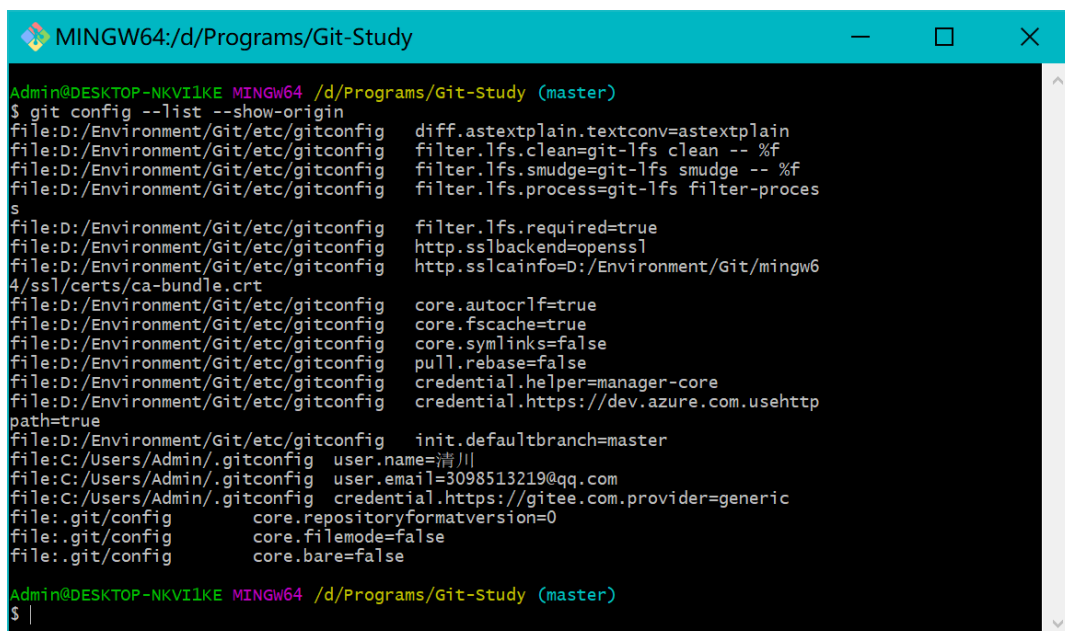
```
1 #配置个人的用户名称和电子邮件地址
2 git config --global user.name "用户名"
3 git config --global user.email "邮箱"
```

再次强调，如果使用了 **--global** 选项，那么该命令只需要运行一次，因为之后无论你在该系统上做任何事情，Git 都会使用那些信息。当你想针对特定项目使用不同的用户名称与邮件地址时，可以在那个项目目录下运行没有 **--global** 选项的命令来配置。

• 查看配置

- 查看所有配置及它们所在的位置

```
1 #查看所有配置及它们所在的文件
2 git config --list --show-origin
3 # :wq 退出查看
```



```
Admin@DESKTOP-NKVILKE MINGW64 /d/Programs/Git-Study (master)
$ git config --list --show-origin
file:D:/Environment/Git/etc/gitconfig diff.astextplain.textconv=astextplain
file:D:/Environment/Git/etc/gitconfig filter.lfs.clean=git-lfs clean -- %f
file:D:/Environment/Git/etc/gitconfig filter.lfs.smudge=git-lfs smudge -- %f
file:D:/Environment/Git/etc/gitconfig filter.lfs.process=git-lfs filter-proces
s
file:D:/Environment/Git/etc/gitconfig filter.lfs.required=true
file:D:/Environment/Git/etc/gitconfig http.sslbackend=openssl
file:D:/Environment/Git/etc/gitconfig http.sslcainfo=D:/Environment/Git/mingw6
4/ssl/certs/ca-bundle.crt
file:D:/Environment/Git/etc/gitconfig core.autocrlf=true
file:D:/Environment/Git/etc/gitconfig core.fscache=true
file:D:/Environment/Git/etc/gitconfig core.symlinks=false
file:D:/Environment/Git/etc/gitconfig pull.rebase=false
file:D:/Environment/Git/etc/gitconfig credential.helper=manager-core
file:D:/Environment/Git/etc/gitconfig credential.https://dev.azure.com.usehttp
path=true
file:D:/Environment/Git/etc/gitconfig init.defaultbranch=master
file:C:/Users/Admin/.gitconfig user.name=清川
file:C:/Users/Admin/.gitconfig user.email=3098513219@qq.com
file:C:/Users/Admin/.gitconfig credential.https://gitee.com.provider=generic
file:./git/config core.repositoryformatversion=0
file:./git/config core.filemode=false
file:./git/config core.bare=false
Admin@DESKTOP-NKVILKE MINGW64 /d/Programs/Git-Study (master)
$ |
```

- 查看配置信息

```
1 #查看配置，下面两个命令都可以
2 git config --list
3 git config -l
4 # :wq 退出查看
```

```
MINGW64:/d/Programs/Git-Study
Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=D:/Environment/git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=清川
user.email=3098513219@qq.com
credential.https://gitee.com.provider=generic
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.gitee.url=https://gitee.com/qingchuancoder/GitStudy.git
remote.gitee.fetch=+refs/heads/*:refs/remotes/Gitee/*

Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
```

- 查看系统配置

```
1 #查看系统配置
2 git config --system --list
3 git config --system -l
```

```
MINGW64:/d/Programs/Git-Study
Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git config --system -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=D:/Environment/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master

Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$
```

- 查看当前用户(global)配置

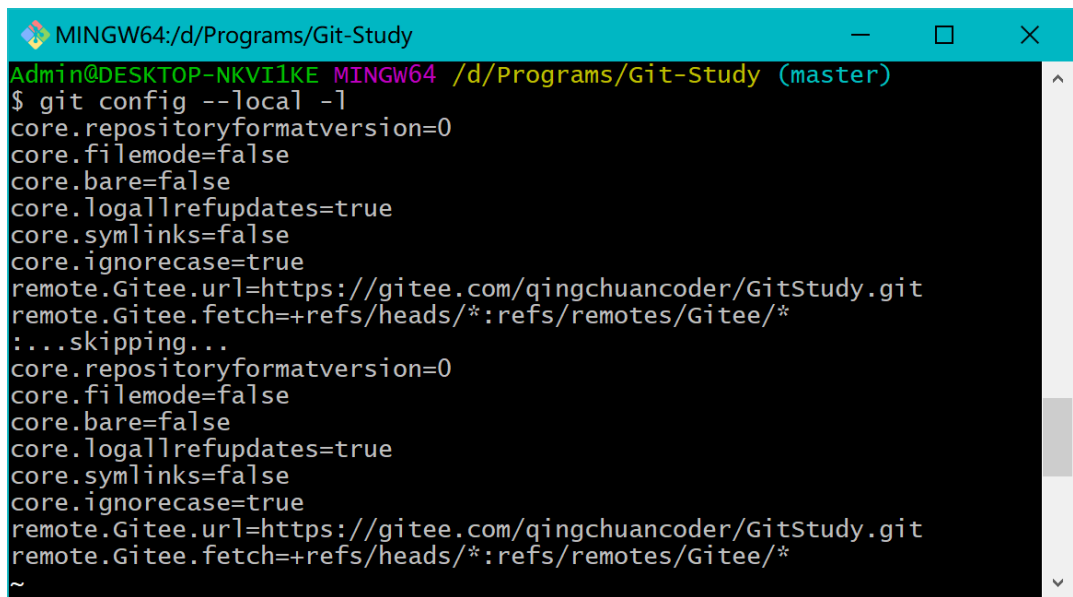
```
1 #查看当前用户(global)配置
2 git config --global --list
3 git config --global -l
```

```
MINGW64:/d/Programs/Git-Study
Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git config --global -l
user.name=清川
user.email=3098513219@qq.com
credential.https://gitee.com.provider=generic

Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$
```

- 查看当前仓库配置

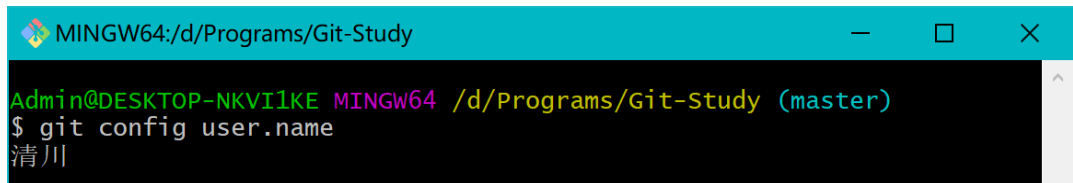
```
1 #查看当前仓库配置
2 git config --local --list
3 git config --local -l
```



```
Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git config --local -l
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.Gitee.url=https://gitee.com/qingchuancoder/GitStudy.git
remote.Gitee.fetch=+refs/heads/*:refs/remotes/Gitee/*
...skipping...
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.Gitee.url=https://gitee.com/qingchuancoder/GitStudy.git
remote.Gitee.fetch=+refs/heads/*:refs/remotes/Gitee/*
~
```

- 查看某一项配置

```
1 #git config 配置项
2 git config user.name
```



```
Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git config user.name
清川
```

查看配置时可能会发现重复的变量名，那是因为Git会从不同的文件中读取同一个配置。
Git 会使用它找到的每一个变量的最后一个配置。

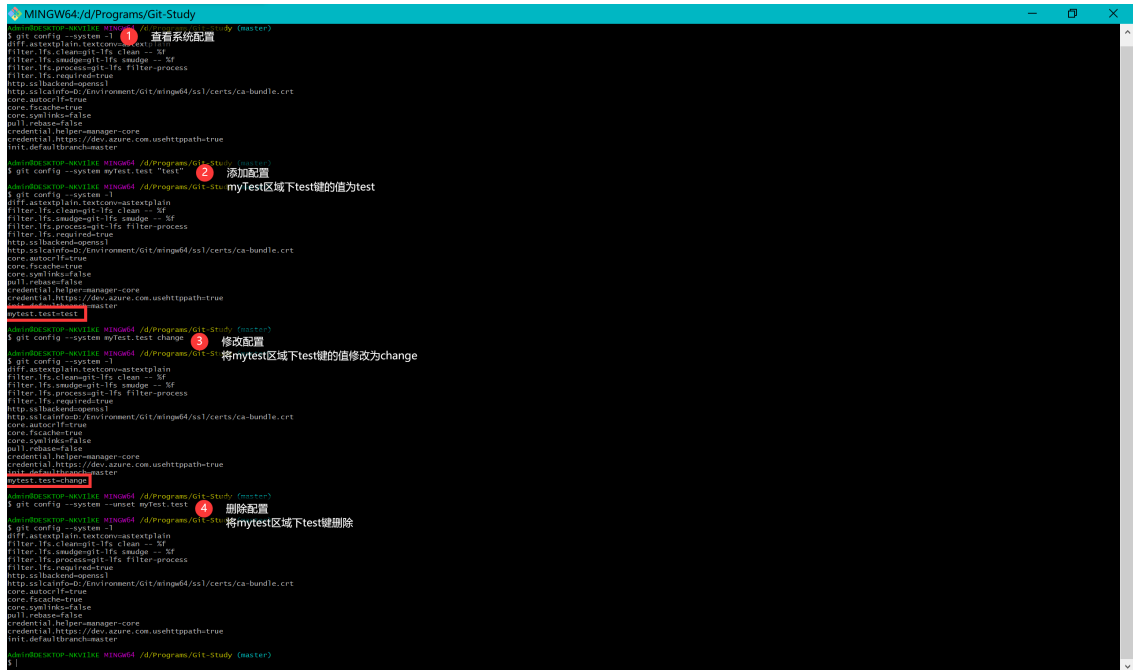
文件	位置
系统级配置文件	git安装目录/etc/gitconfig(D:\Environment\Git\etc\gitconfig)
用户配置文件	~/.gitconfig即C盘用户目录下的gitconfig (C:\Users\Admin\.gitconfig)
仓库配置文件	git项目目录的.git\config(D:\Programs\Git-Study.git\config)

• 添加配置

```
1 #[--local|--global|--system]表示可选，分别对应当前项目、全局、系统不同级别的设置
2 #section.key表示区域下的键
3 #value表示值
4 git config [--local|--global|--system] section.key value
```

删除配置

```
1 git config [--local|--global|--system] --unset section.key
```



```
MINGW64/d/Programs/Git-Study (master)
$ git config --system -l
diff.autoplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcert=/c:/ProgramData/Git/cmd/git/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.sshkey=ssh-keygen
credential.helper=manager-core
credential.https://dev.azure.com.usshelpath=true
init.defaultbranch=master

MINGW64/d/Programs/Git-Study (master)
$ git config --system myTest.test "test"
$ git config --system -l
diff.autoplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcert=/c:/ProgramData/Git/cmd/git/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.sshkey=ssh-keygen
credential.helper=manager-core
credential.https://dev.azure.com.usshelpath=true
init.defaultbranch=master
myTest.test=test

MINGW64/d/Programs/Git-Study (master)
$ git config --system myTest.test change
$ git config --system -l
diff.autoplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcert=/c:/ProgramData/Git/cmd/git/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.sshkey=ssh-keygen
credential.helper=manager-core
credential.https://dev.azure.com.usshelpath=true
init.defaultbranch=master
myTest.test=change

MINGW64/d/Programs/Git-Study (master)
$ git config --system --unset myTest.test
$ git config --system -l
diff.autoplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcert=/c:/ProgramData/Git/cmd/git/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.sshkey=ssh-keygen
credential.helper=manager-core
credential.https://dev.azure.com.usshelpath=true
init.defaultbranch=master
```

至此，关于Git的准备工作差不多完成了，接下来，愉快的使用Git吧！

Git基本操作

获取Git仓库

在已存在的目录中初始化仓库

该命令将创建一个名为 `.git` 的子目录，这个子目录含有你初始化的 Git 仓库中所有的必须文件，这些文件是 Git 仓库的骨干。但是，在这个时候，我们仅仅是做了一个初始化的操作，你的项目里的文件还没有被跟踪。

克隆现有的仓库

如果你想获得一份已经存在了的 Git 仓库的拷贝，比如说，你想为某个开源项目贡献自己的一份力，这时就要用到 `git clone` 命令。

个人理解：

也就是说，`git init` 是在本地初始化仓库；`git clone [url]` 是将远程的仓库拷贝到本地。

两者的差别是，`git init` 无需提前在远程存在一个仓库，但是这样初始化的仓库并没有和远程仓库联系；`git clone [url]` 克隆的仓库需要提前创建一个远程仓库，不过克隆过后本地的仓库就和远程的仓库联系起来了。

推荐现在远程创建好一个仓库再克隆到本地

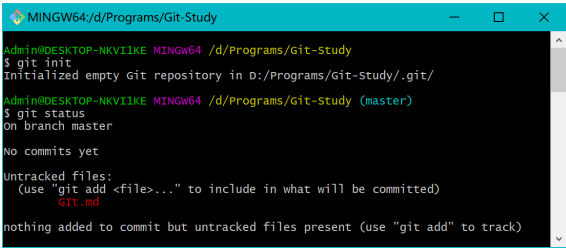
命令	说明	
git init	初始化本地仓库	
git clone [url]	克隆。如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：git clone git://github.com/schacon/grit.git mygrit	

记录每次更新到仓库

工作目录下的文件状态：**已跟踪** 或 **未跟踪**。

文件在Git 内都只有三种状态：**已提交 (committed)**，**已修改 (modified)** 和**已暂存 (staged)**。

检查当前文件状态

命令	说明	
git status	查看所有文件状态	
git status [filename]	查看指定文件的状态	

跟踪新文件

命令	说明
git add files	将文件从工作目录添加到暂存区；git add . 添加所有文件

此时再运行 `git status` 命令，会看到文件已被跟踪，并处于暂存状态

暂存已修改文件

将已跟踪文件修改后，该文件会变成已修改 (modified) 状态，并且还没放到暂存区。

要暂存这次更新，需要运行 `git add` 命令。

`git add`这是个多功能命令：可以用它开始跟踪新文件，或者把已跟踪的文件放到暂存区，还能用于合并时把有冲突的文件标记为已解决状态等。将这个命令理解为“精确地将内容添加到下一次提交中”而不是“将一个文件添加到项目中”要更加合适。

状态简览

```
1 #以简洁的形式查看状态
2 git status --short
3 git status -s
```

- 新添加的未跟踪文件前面有 `??` 标记
- 新添加到暂存区中的文件前面有 `A` 标记 (add)
- 修改过的文件前面有 `M` 标记 (modified)
- 输出中有两栏，左栏指明了暂存区的状态，右栏指明了工作区的状态。

示例：

```
1 $ git status -s
2  M README
3  MM Rakefile
4  A lib/git.rb
5  M lib/simplegit.rb
6  ?? LICENSE.txt
```

状态报告显示：

- `README` 文件在工作区已修改但尚未暂存
- 而 `lib/simplegit.rb` 文件已修改且已暂存
- `Rakefile` 文件已修，暂存后又作了修改，因此该文件的修改中既有已暂存的部分，又有未暂存的部分。

忽略文件

我们可以创建一个名为 `.gitignore` 的文件，列出要忽略的文件模式。

```
1 *.text      #忽略所有.text结尾的文件
2 !lib.text   #lib.text除外
3 /temp       #仅忽略根目录下的temp目录
4 build/      #忽略build/目录下所有的文件
```

文件 `.gitignore` 的格式规范如下：

- 所有空行或者以注释符号 `#` 开头的行都会被 Git 忽略。
- 可以使用标准的 glob 模式匹配。
- 匹配模式最后跟反斜杠 (`/`) 说明要忽略的是目录。
- 要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号 (`!`) 取反。

查看已暂存和未暂存的修改

```
1 #比较工作目录中当前文件和暂存区快照间的差异
2 git diff
3
4 #比较已暂存文件和最后一次提交的文件差异
5 git diff --staged
```

提交更新

现在的暂存区已经准备就绪，可以提交了。在此之前，请务必确认还有什么已修改或新建的文件还没有 `git add` 过，否则提交的时候不会记录这些尚未暂存的变化。这些已修改但未暂存的文件只会保留在本地磁盘。所以，每次准备提交前，先用 `git status` 看下，你所需要的文件是不是都已暂存起来了，然后再运行提交命令 `git commit`

```
1 #git commit -m "提交了Git.md"    将文件提交到本地仓库
2 git commit -m "消息"
```

跳过使用暂存区

```
1 #把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤
2 git commit -a -m "跳过暂存区，直接提交"
```

这很方便，但是要小心，有时这个选项会将不需要的文件添加到提交中。

移除文件

- 要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。
- 可以用 `git rm` 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。
- 如果只是简单地从工作目录中手工删除文件，运行 `git status` 时就会在“Changes not staged for commit”部分（也就是 未暂存清单）看到。
- 如果要删除之前修改过或已经放到暂存区的文件，则必须使用强制删除选项 `-f`（译注：即 force 的首字母）。这是一种安全特性，用于防止误删尚未添加到快照的数据，这样的数据不能被 Git 恢复。
- 另外一种情况是，我们想把文件从 Git 仓库中删除（亦即从暂存区域移除），但仍然希望保留在当前工作目录中。`git rm --cached [filename]`

个人理解：

- `git rm [filename]` 从跟踪列表删除并删除工作目录下的文件
- 手工删除 工作目录下的文件会被删除，但Git仍在跟踪，需要使用git rm命令将其从跟踪列表删除，在提交后，该文件将不再出现在列表
- `git rm -f [filename]` 删除之前修改过或已暂存的文件
- `git rm --cached [filename]` 从Git仓库中删除，但仍然保留在工作目录

重命名文件

```
mv [filename] [newname]
```

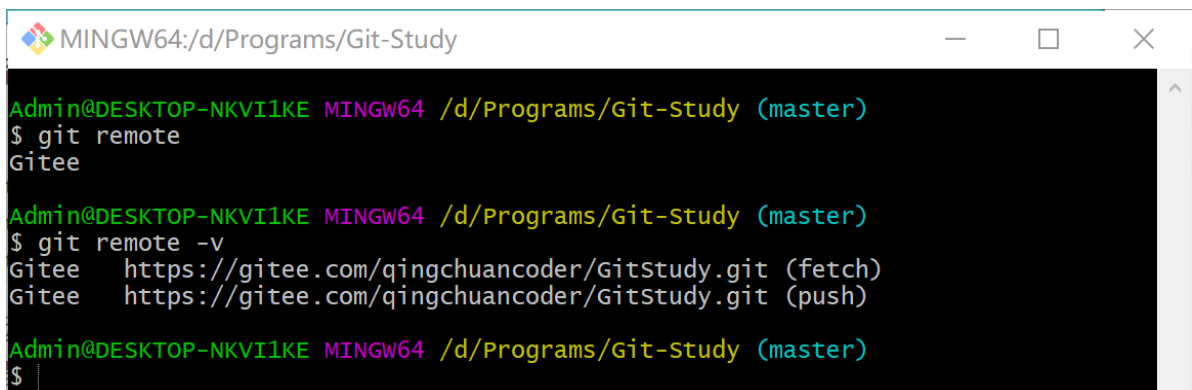
撤销操作

```
1 #撤销提交
2 git commit --amend
3
4 #取消暂存
5 git restore --staged [filename]
6
7 #取消修改
8 git restore [filename]
9
10 #刚刚新建的文件add后unstage
11 git rm --cached [file]...
```

远程仓库的使用

查看远程仓库

```
1 #查看远程仓库
2 git remote
3
4 #查看远程仓库及URL
5 git remote -v
```



```
MINGW64:/d/Programs/Git-Study
Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git remote
Gitee

Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$ git remote -v
Gitee https://gitee.com/qingchuancoder/GitStudy.git (fetch)
Gitee https://gitee.com/qingchuancoder/GitStudy.git (push)

Admin@DESKTOP-NKVI1KE MINGW64 /d/Programs/Git-Study (master)
$
```

添加远程仓库

```
1 #shortname 是仓库的名字， url是仓库的地址
2 git remote add [shortname] [url]
```

从远程仓库抓取与拉取

```
1 #拉去远程仓库的数据
2 git fetch [remote]
3
4 #自动抓取后合并该远程分支到当前分支
5 git pull
```

必须注意 `git fetch` 命令只会将数据下载到你的本地仓库——它并不会自动合并或修改你当前的工作。

推送到远程仓库

```
1 #将branch分支推送到remote 服务器
2 git push [remote] [branch]
```

查看某个远程仓库

```
1 #remote是仓库名
2 git remote show [remote]
```

远程仓库的重命名与移除

```
1 #重命名
2 git remote rename "原名" "新名"
3
4 #移除仓库,remote是仓库名
5 git remote remove [remote]
6 git remote rm [remote]
```

SSH

1. 生成

```
1 ssh-keygen -t rsa
```

敲三次回车即可

2. 在Gitee中使用

标签

列出标签

```
1 #以字母顺序列出标签
2 git tag
3
4 #也可以添加选项
5 git tag -l
6
7 #可以利用通配符，但是此时必须使用-l选项
8 git tag -l "v1.8.5*"
```

创建标签

Git 支持两种标签：轻量标签（lightweight）与附注标签（annotated）。

轻量标签很像一个不会改变的分支——它只是某个特定提交的引用。

而附注标签是存储在 Git 数据库中的一个完整对象，它们是可以被校验的，其中包含打标签者的名字、电子邮件地址、日期时间，此外还有一个标签信息，并且可以使用 GNU Privacy Guard（GPG）签名并验证。通常会建议创建附注标签，这样你可以拥有以上所有信息。但是如果你只是想用一个临时的标签，或者因为某些原因不想要保存这些信息，那么也可以用轻量标签。

附注标签

```
1 git tag -a v1.4 -m "my version 1.4"
```

轻量标签

```
1 git tag v1.4-lw
```

后期打标签

```
1 #v1.2是标签 9fceb02是提交的校验和（部分校验和）
2 git tag -a v1.2 9fceb02
```

删除标签

```
1 #tagname是标签名
2 git tag -d [tagname]
```

Git别名

```
1 #checkout=>co
2 git config --global alias.co checkout
3
4 #commit=>ci
5 git config --global alias.ci commit
```

Git分支

```
1 #列出所有分支
2 git branch
3
4 #查看每一个分支的最后一次提交
5 git branch -v
6
7 #查看哪些分支已经合并到当前分支
8 #此列表中没有*号的分支通常可以删除掉，因为已经将他们的工作合并到了一个分支
9 git branch --merged
10
11 #查看所有包含未合并工作的分支
12 git branch --no-merged
13 # 查看合并与未合并也可以添加分支名来查看指定分支的
14
15 #列出所有远程分支
16 git branch -r
17
18 #新建一个分支，但依然停留在当前分支
19 git branch [branchname]
20
21 #切换到一个已存在分支
22 git checkout [branchname]
23
24 #新建一个分支，并切换到该分支
25 git checkout -b [branch]
26
27 #合并分支到当前分支
28 #将branch分支合并到当前这个分支
29 git merge [branch]
30
31 #删除分支
32 git branch -d [branch-name]
33
34 #删除远程分支
35 git push origin --delete [branch-name]
36 git branch -dr [remote/b]
```

